(2)

Technical Report 1440
July 1991

Matched Field Processing on the Connection Machine

T. A. Adams

NAVAL OCEAN SYSTEMS CENTER

San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commander

R. T. SHEARER, Acting
Technical Director

ADMINISTRATIVE INFORMATION

The work reported here was performed by members of the Signal Processing Technology Branch (Code 733), Signal and Information Processing Division, Surveillance Department, with funding provided by the Office of Naval Technology (Code 227), 800 N. Quincy, Arlington, VA 22217, under Computer Technology Block (N02D).

Released by
D. K. Barbour, Head
Signal Processing
Technology Branch

Under authority of
J. A. Roese, Head
Signal and Information
Processing Division

ACKNOWLEDGMENT

The author is grateful to Paul Hertz of the Naval Research Laboratory (Code 4121.5) for providing the fast Fourier transform software for the Connection Machine.

RII

CONTENTS

FIGURES

1.0 INTRODUCTION

In the search for improved detection performance in sonar signal processing, there has been a trend toward the use of more complex processing methods. An interesting example is matched field processing, in which the assumptions of plane wave propagation are discarded in favor of more detailed models of ocean acoustics. The extra detection performance of these methods is achieved at the expense of additional computational effort. However, the increasing availability of parallel computers motivates us to explore the application of these new machines to challenging problems of sonar signal processing.

This report discusses work performed to implement matched-field processing on the Thinking Machines Corporation's Connection Machine (model CM-2). This was part of a task with twofold objectives. One was to develop a high-performance computing capability for the specific matched field processing application. The other was to advance generic software technology, specifically to address the difficult issue of software portability for parallel machines. In this report, the discussion will be focused primarily on the former objective.

2.0 MATCHED FIELD PROCESSING

Many future undersea surveillance systems are likely to incorporate some form of the signal processing technique known as matched field processing (MFP). The essence of the method is depicted in figures 1 through 3. Output power indicates the degree of match between measured sound pressure fields (from sensor data) and model predictions (from replica data). The output power is to be computed for a multitude of ranges, azimuths, depths, and frequencies. An important observation is that matched field processing has, to varying degrees along the processing chain, high levels of parallelism in the frequency, spatial location, and sensor dimensions. For example, FFTs can be computed in parallel for all sensors; each FFT has further levels of exploitable parallelism (i.e., individual butterfly computations).

There are a number of variants of matched field processing. In this task, it was initially planned to implement four different forms of matched field processing, referred to as subsampled MVDR, full MVDR, conventional MFP, and array partitioning. The most general form of these four is array partitioning, which is the method shown in figure 1. (Array partitioning is described in more detail in the appendix.) By performing the quadratic forms part of the computation in different ways, either Bartlett processing or minimum variance distortionless response (MVDR) processing can be considered. MVDR is also known as the maximum likelihood method. These two alternatives for the quadratic forms are discussed in [Baggeroer, et al., 1988]. Subsampled MVDR and full MVDR are specializations in which the spatial filtering and summation over subarray is bypassed. Subsampled MVDR and full MVDR are actually the same algorithm with different implementation details on a moderately parallel machine (subsampled MVDR would perform matrix algebra computations without interprocessor communication; full MVDR would employ interprocessor communication; the distinction between subsampled and full disappears on the Connection Machine). The MVDR processing chain is shown in figure 2. Conventional MFP is the further specialization in which Bartlett processing takes the place of the minimum variance computations; that is, the subarray matrix factoring is bypassed. This is shown in figure 3.

Each method has its own advantages and disadvantages. Conventional MFP is the simplest and was implemented on the Connection Machine (apart from the computation of the narrowband time series). MVDR is somewhat more complicated and computationally expensive than conventional MFP, but yields better detection performance. Array partitioning is the most complicated, but has the potential to yield much better detection performance for a given level of computational effort.

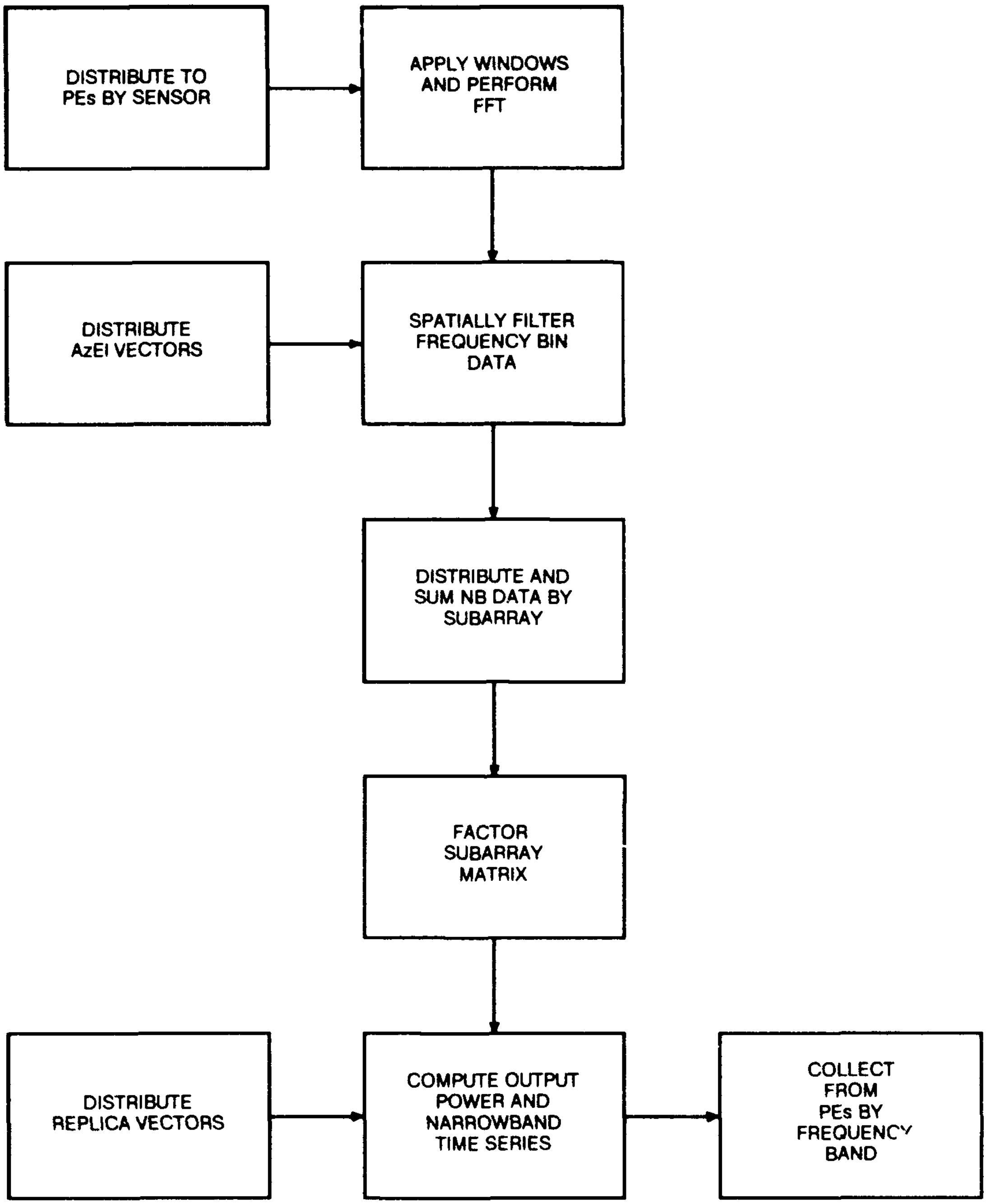

Figure 1. Matched-field processing with array partitioning.

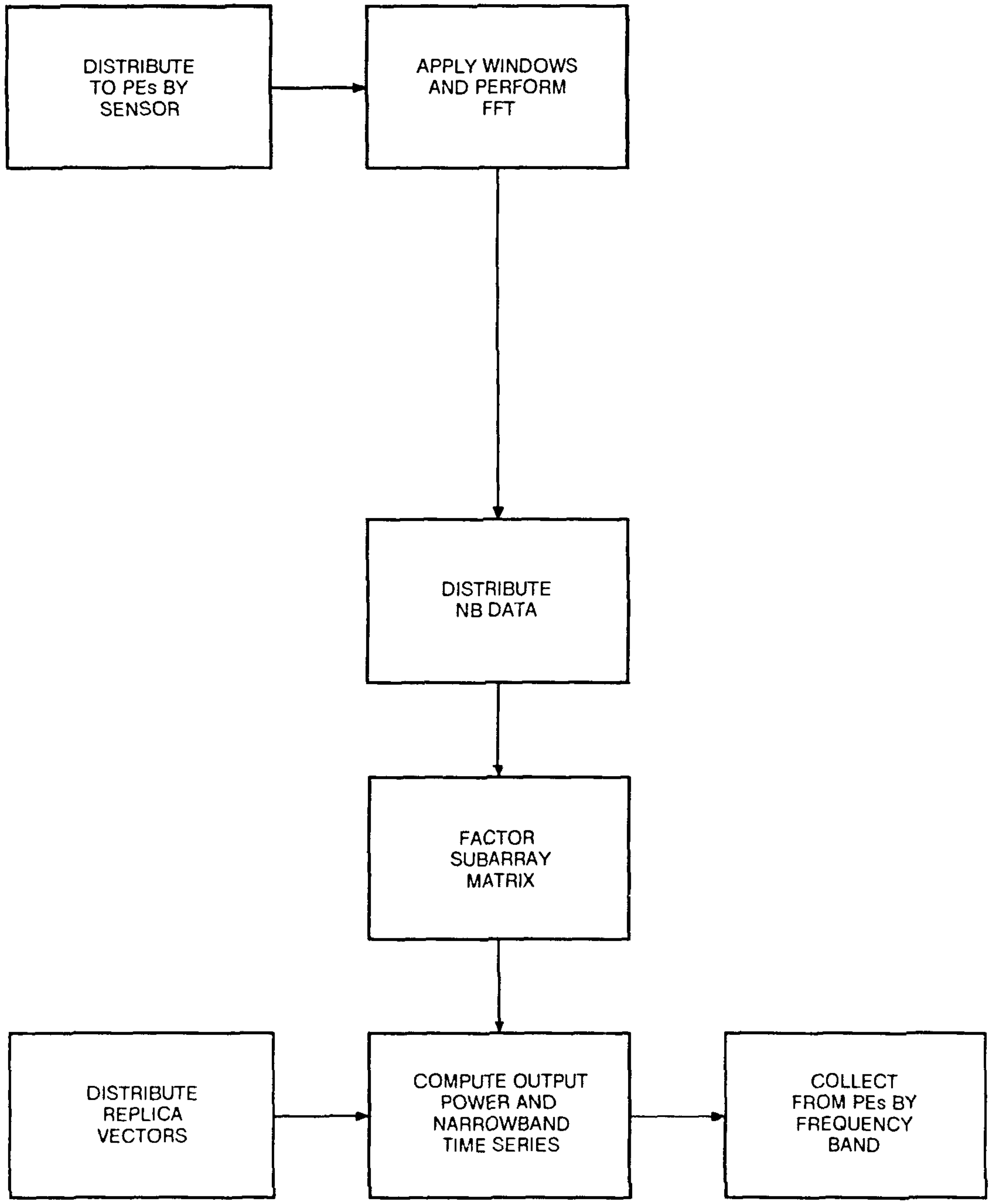

Figure 2. Minimum variance distortionless response processing.

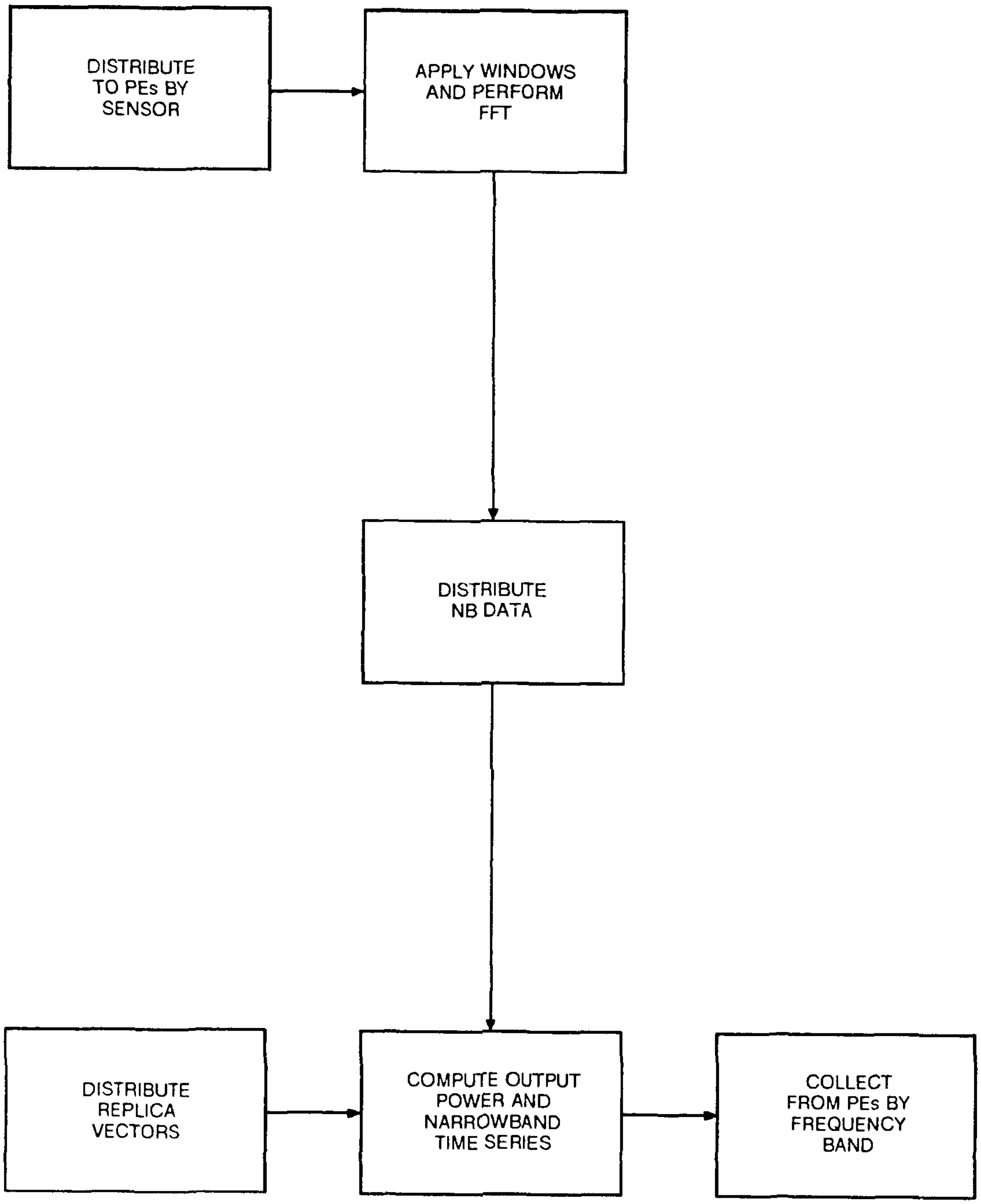

Figure 3. Conventional matched-field processing.

3.0 TARGET HARDWARE

A Connection Machine contains thousands of bit-serial processors arranged in groups of 32. Each group of 32 physical processors consists of two custom (CM) chips, each containing 16 physical processors, together with a memory chip, a floating-point accelerator chip, and a chip to interface the floating-point accelerator with the memory. By means of a time-slicing technique, each physical processor can perform virtual processing; in other words, the Connection Machine can be operated to appear transparently to have a larger number of physical processors than it actually has. The ratio of virtual processors to physical processors is referred to as the virtual processor ratio (VPR). In general, it is advantageous to be able to use high VPR, since this leads to more efficient processing. Processors can communicate with one another either through the router, which allows any processor to communicate with any other processor, or through the north-east-west-south (NEWS) grid, which permits communication over an N-dimensional rectangular mesh. An important observation is that the communications expense is highly dependent on whether the router or the NEWS grid is used, and on whether the communications are intragroup or intrachip. This has important implications for the way the data structures of the algorithm should be arranged over the processors of the CM-2. The activities of the Connection Machine are coordinated by a conventional sequential computer known as the front end.

Other noteworthy features of the Connection Machine are the data vault, a disk-array-based mass storage device, and the framebuffer, a high-resolution graphics display. Both of these facilities make use of the parallel processing features of the CM-2 to achieve data transfer at high rates. It is natural to exploit parallelism in the I/O as well as in the numerical computations, and this was an important element of the work.

The near-term preferred target machine for this effort was the AT&T DSP3, a moderately parallel multiple-instruction-stream multiple-data-stream (MIMD) machine [Shively, et al., 1989]. The immediate matched-field processing requirements were to treat problems with tens to hundreds of sensors and up to tens of frequencies, which appeared to be well suited to the 128 processors of the DSP3. Because the DSP3 is an MIMD machine, it affords the opportunity to work on different parts of the processing chain concurrently. Because the DSP3 was not available at the start of the effort, the Connection Machine (CM-2) from Thinking Machines Corporation, a massively parallel single-instruction-stream multiple-data-stream (SIMD) machine was used initially. The configurations of the CM-2 that were available for this task had 4096, 8192, and 16,384 processors. One of the benefits of using the CM-2 was that its very different architecture and programming environment provided an expanded base of experience useful for later addressing software portability issues. Detailed discussions of the Connection Machine are found in [Hillis, 1985] and [**cm2tecsum**].

4.0 SOFTWARE PORTABILITY FOR PARALLEL PROCESSORS

The initial attempt at addressing the portability issue was to employ a conventional approach of phased development to separate the requirements and high-level design from implementation details. Functional descriptions of the matched field processing algorithms were prepared and reviewed. Code was then written from these functional descriptions. Intermingling of front-end data structures and code with parallel processor data structures and code was kept to a minimum. The front-end data structures and code were written in the C language, while the parallel processor data structures and code were expressed in C*, an extension of C developed for the Connection Machine. Similarly, processes dealing only with interprocessor communication were separated from processes involving numerical computations. From the functional descriptions were derived requirements specifications in DOD–STD–2167A format **[mvdrsrs]**, **[apasrs]** and pseudocode documents **[mvdrpseu]**, **[apapseu]** to facilitate future software development.

The approach described above has severe limitations. A key difficulty is that the "distance" or dissimilarity between the code and a relatively machine-independent intermediate representation (e.g., pseudocode) is great. Consequently, the effort in translating from a high-level representation to code is substantial and this effort must still be expended anew with each new machine.

5.0 MAPPING ONTO THE CONNECTION MACHINE

The CM-2 source code for the conventional MFP appears in [**apascl**].

Matched-field processing (as well as similar signal-processing algorithms) consists of a chain of processes with the outputs of one process serving as the inputs to the next process in the chain. A massively parallel implementation of each process involves the use of N-dimensional rectangular meshes over which the data are arranged for the parallel computations, with different meshes (including different values of N) being appropriate for the different processes of the algorithm and different stages within a process. In the array partitioning algorithm, the processes are **distribute to PEs by sensor, apply windows and perform FFT, distribute AzEl vectors, spatially filter frequency bin data, distribute and sum NB data by subarray** (called **distribute NB data** in the non-array-partitioning case), **factor subarray matrix, distribute replica vectors, compute output power and narrowband time series**, and **collect from PEs by frequency band**. These are discussed in the appendix. The subset capability implemented on the Connection Machine consisted of conventional MFP only, with no computation of the narrowband time series. The processes associated with this subset capability are **distribute to PEs by sensor, apply windows and perform FFT, distribute NB data, distribute replica vectors, compute output power**, and **collect from PEs by frequency band.**

The rectangular mesh associated with a particular process reflects the parallelism inherent in that process. For example, in the **apply windows and perform FFT** process, the data are naturally arranged over a two-dimensional mesh, with the dimensions corresponding to sensor and time on input and sensor and frequency on output. It is also important to note the dimensions with respect to which the computations are totally decoupled or "embarrassingly parallel" (EP). For example, in the **apply windows and perform FFT** process, all sensor channels can be treated completely independently of one another, so we say the process is EP with respect to sensor. The rectangular meshes are indicated in figure 4, with the EP mesh edges indicated in upper case; the labeling applies at the conclusion of each process' execution. By identifying the dimensions over which the processing is EP, it is possible to decide how to arrange data over the processors to keep the communications costs low. The parallelism of our Connection Machine implementation of conventional MFP is shown in figure 5. Real-world limitations such as finite memory prevent us from exploiting all the intrinsic parallelism of an ideal algorithm.

It should be noted that the implementation of the software discussed in this report does not exploit the "EP-ness" of the problem in this way because the software uses the less efficient router communications only. However, it should not be too difficult to rewrite the software to use the more efficient N-d grid package (from the NRL C* library), which employs the NEWS grid to perform fast nearest-neighbor

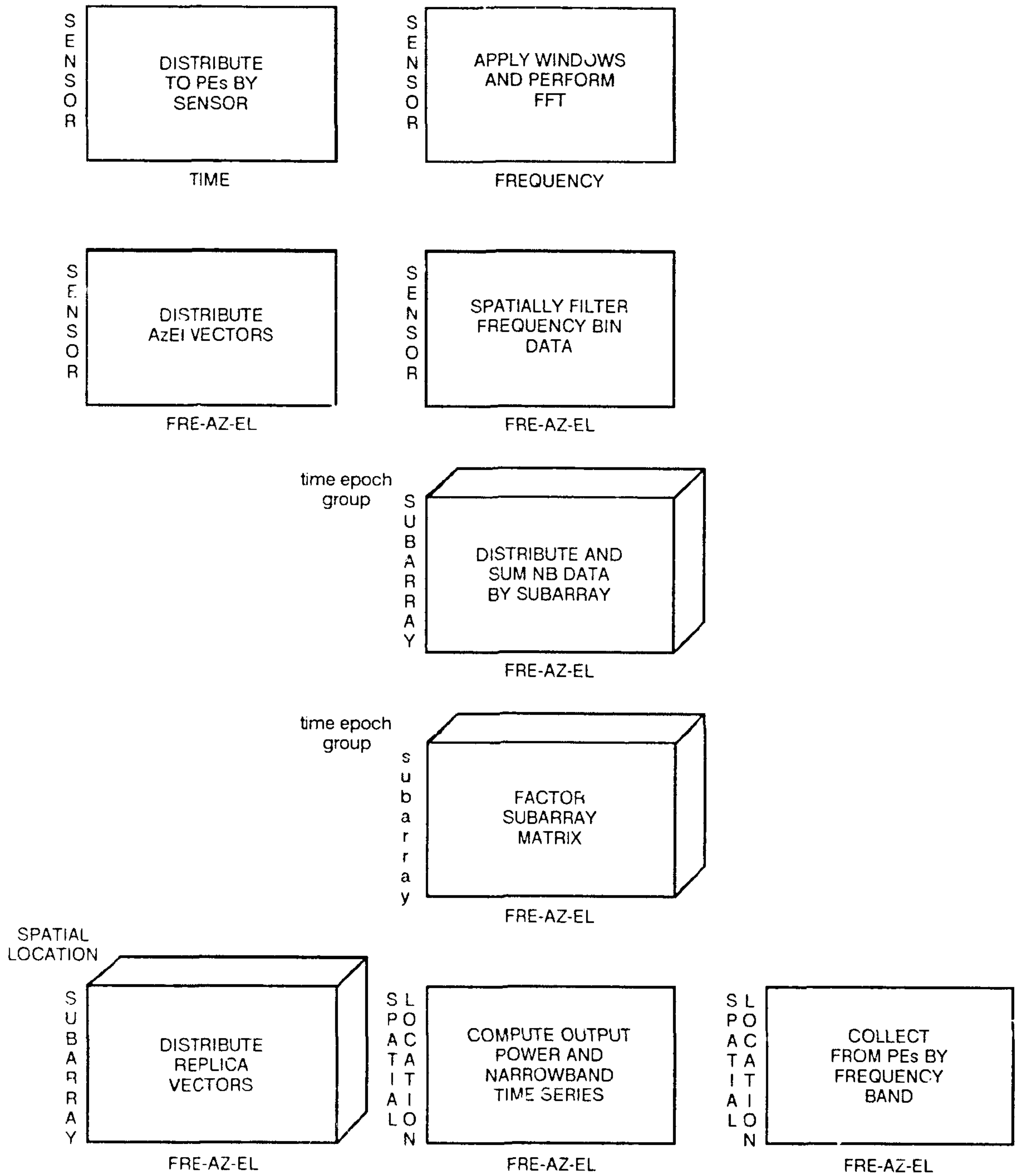

Figure 4. Parallelism in the array-partitioning processing chain.

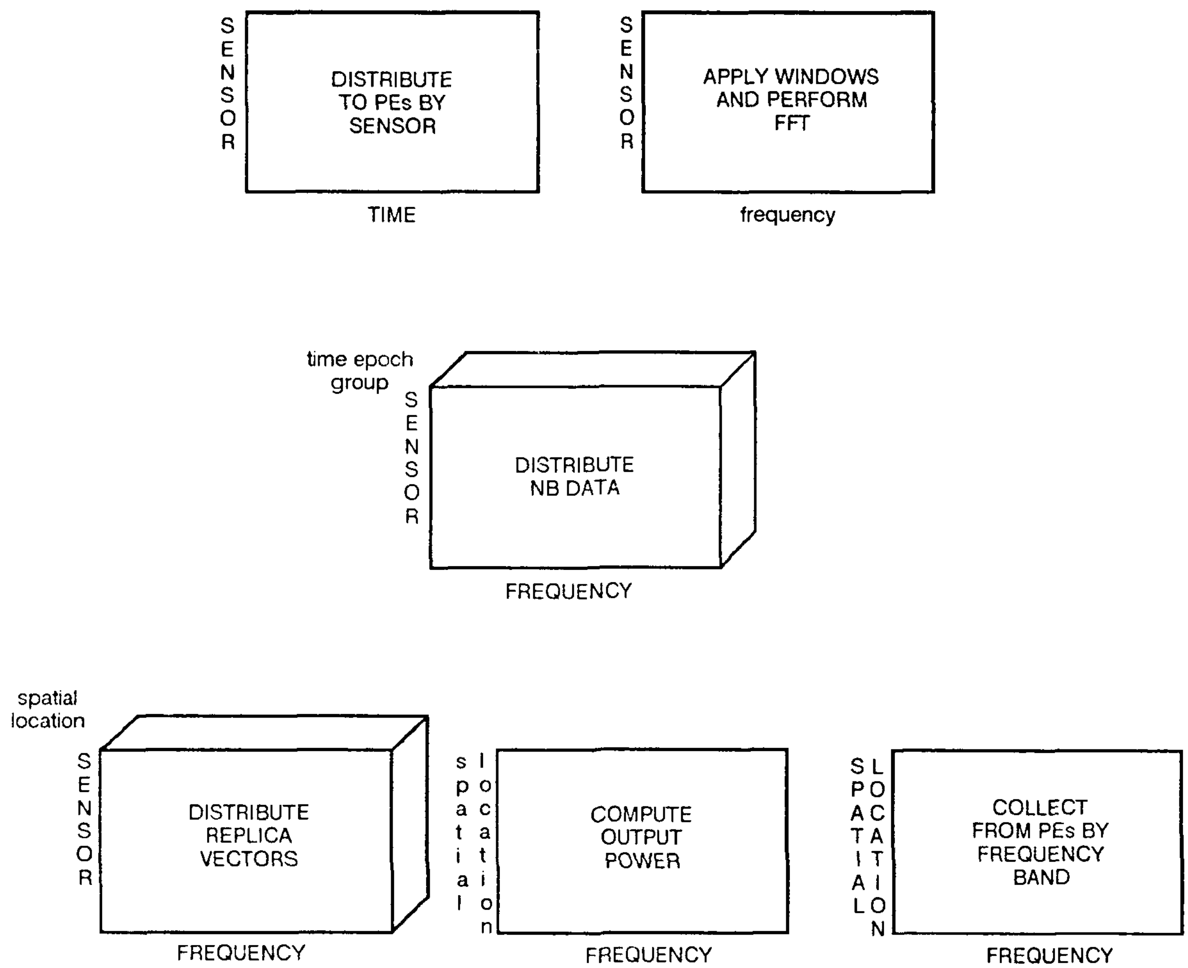

Figure 5. Parallelism in the Connection Machine implementation of conventional MFP.

communications. Some extensions to the N–d grid package would be needed to exploit the fact that communications are low or nonexistent along certain mesh dimensions. It would still be necessary to use router communications in some parts of the algorithm.

The **distribute to PEs by sensor** process uses a sensor/time 2-D mesh, and is EP with respect to sensor and time.

The **apply windows and perform FFT** process uses a sensor/time 2-D mesh (input) and sensor/frequency 2-D mesh (output), and is EP with respect to sensor.

The **distribute AzEl vectors** process uses a sensor/frequency 2-D mesh (input) and sensor/frequency-azimuth-elevation 2-D mesh (output), and is EP with respect to sensor and frequency-azimuth-elevation. This was not a part of our Connection Machine implementation.

The **spatially filter frequency bin data** process uses a sensor/frequency-azimuth-elevation 2-D mesh, and is EP with respect to sensor and frequency-azimuth-elevation. This was not a part of our Connection Machine implementation.

The **distribute and sum NB data by subarray** process uses a sensor/frequency-azimuth-elevation 2-D mesh (input) and a subarray/frequency-azimuth-elevation/time epoch group 3-D mesh (output), and is EP with respect to subarray and frequency-azimuth-elevation. Our Connection Machine implementation (of **distribute NB data**) is EP with respect to frequency only.

The **factor subarray matrix** process uses a subarray/frequency-azimuth-elevation/ time epoch group 3-D mesh, and is EP with respect to frequency-azimuth-elevation. This was not a part of our Connection Machine implementation.

The **distribute replica vectors** process uses a subarray/frequency-azimuth-elevation/ spatial location 3-D mesh, and is EP with respect to subarray, spatial location, and frequency-azimuth-elevation. Our Connection Machine implementation is EP with respect to sensor and frequency only.

The **compute output power and narrowband time series** process uses a subarray/ (spatial location or time epoch) column group/frequency-azimuth elevation 3-D mesh (input) and a spatial location/frequency-azimuth-elevation 2-D mesh (output), and is EP with respect to spatial location and frequency-azimuth-elevation. Our Connection Machine implementation was EP with respect to frequency only.

The **collect from PEs by frequency band** process uses a spatial location/frequency-azimuth-elevation 2-D mesh, and is EP with respect to spatial location and frequency-azimuth-elevation.

Note that downstream of the **apply windows and perform FFT** process, the entire processing (sub)chain is EP with respect to frequency-azimuth-elevation.

6.0 VISUALIZATION FOR MATCHED-FIELD PROCESSING

The process **collect from PEs by frequency band** produces a large volume of output data, indexed by spatial location (range and depth), frequency, and time epoch. Because matched-field processing is a relatively unexplored area of investigation, it is worthwhile to be able to present the output data to an analyst with little data reduction so as to foster the insights needed for subsequent, more structured statistical analyses. For example, prior to attempting an empirical probability of detection analysis, it is necessary to have a reasonably good *a priori* knowledge of a target's location in range and depth, a task that is made difficult by the ambiguities introduced by the complicated propagation of sound in the ocean.

An approach to presenting the kind of multidimensional data set used in this task was to employ the Connection Machine's framebuffer to rapidly play back outputs stored on the data vault for many time epochs as an animation or "movie." Such a movie consists of a series of frames appearing on the display in rapid succession. Each frame consists of a collection of B-scan displays, each one corresponding to a different frequency. Each B-scan display indicates output power as gray level (as a function of range and depth). This is illustrated in figure 6.

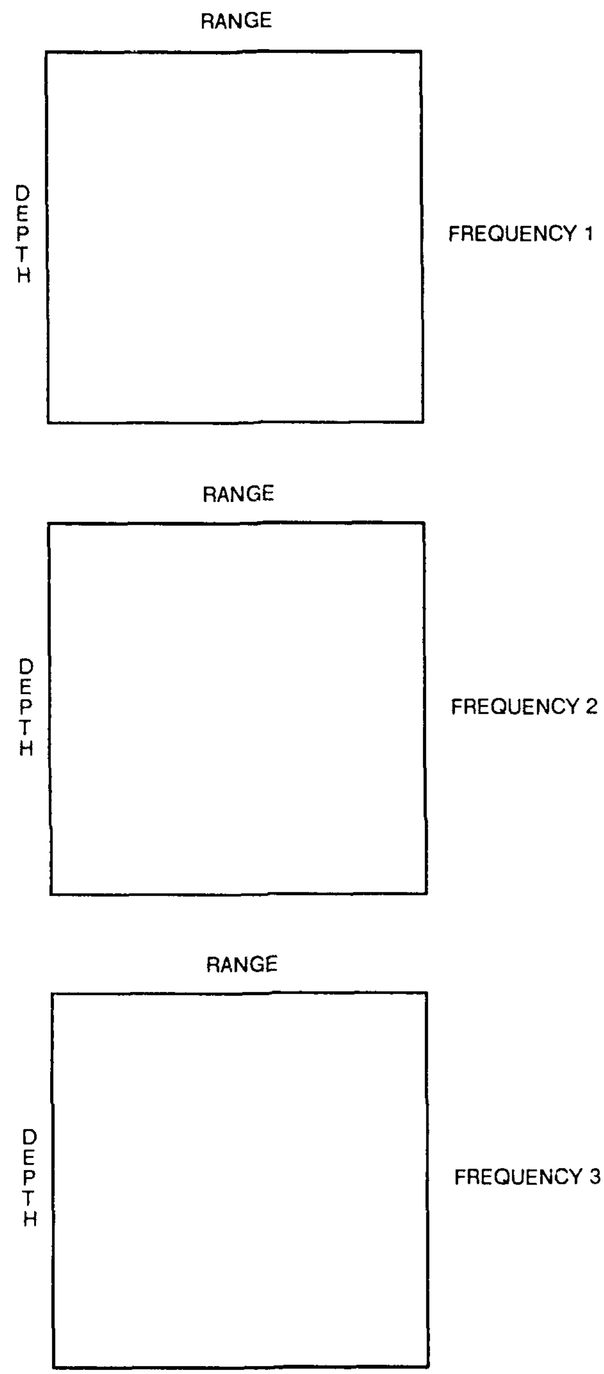

Figure 6. Frame format for visualization of matched-field processor output.

7.0 PRELIMINARY PERFORMANCE EVALUATION

A rudimentary evaluation of the implementation of conventional MFP on the Connection Machine was done to gauge the performance, at least in order-of-magnitude terms. The parameters of the test case were as follows: 4096 spatial locations, 32 sensors, 8 retained frequency bins, and one epoch comprising 256 temporal points per FFT window. This test case was evaluated by using a CM-2 with 8192 physical processors. The elapsed time for this processing was approximately 8 minutes. Roughly three quarters of this time was consumed in the output power computation, with most of the remainder arising from I/O. Subsequent analysis suggested that this extremely poor performance resulted from the heavy use of router communication.

8.0 CONCLUSIONS AND RECOMMENDATIONS

The effort described in this report pointed up a number of opportunities and difficulties associated with implementing matched-field processing and similar types of sonar signal processing on massively parallel computers.

Conventional MFP was implemented on the Connection Machine. In this initial implementation, the potential of the CM-2 was not realized because the programming style and language features used led to a large interprocessor communications burden.

In subsequent efforts at developing signal processing on parallel processors, there should be additional emphasis on decomposing the overall processing into a relatively small set of building blocks that are of higher level than elementary arithmetic operations on scalars. Broad categories of these low-level building blocks would include (i) matrix operations such as those of the basic linear algebra subprograms; (ii) the fast Fourier transform; (iii) data motion primitives to support such non-numeric operations as buffering with overlap, transpose, gather/scatter, and others.

One of the issues that complicates the development of portable parallel libraries is deciding on appropriate arrangements of data structures over distributed memory. For conventional machines, such matters as row or column ordering and strides are of concern. For parallel computers, the characteristics of the machine play a more substantial role and introduce a larger range of choices that must be made.

It is encouraging to observe that the array partitioning version of matched-field processing has a high degree of exploitable parallelism, with the bulk of the algorithm embarrassingly parallel with respect to frequency-azimuth-elevation. It is also worth noting that the processing of data from external sources is not the only situation in which massively parallel machines and algorithms are relevant. When detailed simulation studies are to be performed, there is an additional problem dimension introduced, namely the realizations of the pseudorandom sequences used to generate databases of output statistics. In this case, we have parallelism with respect to frequency-azimuth-elevation realization. The applicability of massively parallel computers to these simulation studies should be explored.

Some initial explorations were made in visualizing matched field processing for the case of no azimuthal resolution. Introducing the additional dimension of azimuth will provide new challenges.

9.0 REFERENCES

Baggeroer, A. B., W. A. Kuperman, and H. Schmidt. February 1988. "Matched Field Processing: Source Localization in Correlated Noise as an Optimum Parameter Estimation Problem," *J. Acoust. Soc. Am.*, vol. 83, no. 2, pp. 571–587.

Shively, R. R., E. B. Morgan, T. W. Copley, and A. L. Gorin. November 1989. "A High Performance Reconfigurable Parallel Processing Architecture," *Proceedings Supercomputing '89*, IEEE/ACM, Reno, NV, pp. 505–509.

Hillis, W. Daniel. 1985. *The Connection Machine*, MIT Press, Cambridge, MA.

Thinking Machines Corporation. April 1987. "Connection Machine Model CM-2 Technical Summary," Technical Report HA87-4. (TMC, 245 First St., Cambridge, MA 02142–1264.) **[cm2tecsum]**

Science Applications International Corporation. March 1990. "Software Requirements Specification for the Minimum Variance Distortionless Response (MVDR) Algorithm," MVDR-SRS-01-U R0C0 (draft).[1] **[mvdrsrs]**

Science Applications International Corporation. March 1990. "Software Requirements Specification for the Array Partitioning Algorithm (APA)," APA-SRS-01-U R0C0 (draft).[1] **[apasrs]**

Science Applications International Corporation. March 1990. "Pseudocode for the Minimum Variance Distortionless Response (MVDR) Algorithm," MVDR-PCD-01-U R0C0 (draft).[1] **[mvdrpseu]**

Science Applications International Corporation. March 1990. "Pseudocode for the Array Partitioning Algorithm (APA)," APA-PCD-01-U R0C0 (draft).[1] (SAIC, 10260 Campus Point Dr., San Diego, CA 92121.) **[apapseu]**

Science Applications International Corporation. January 1990. "Source Code Listings for the Array Partitioning Algorithm (APA)," APA-SCL-01-U R0C0 (draft).[1] **[apascl]**

[1] The SAIC documents cited here were produced under Navy Contract N66001-87-D-0039, and are available to qualified requesters. For further information, contact the author of this report.

APPENDIX

A.1.0 Partitioned Array Bartlett Program

This is a functional description for a program which forms Bartlett azimuth and elevation beams for each subarray of a partitioned array, then Bartlett or Minimum Variance Distortionless Response (MVDR) Matched Field Processing (MFP) to combine the subarray outputs.

A.1.1 Partitioned Array Bartlett Program Inputs

Raw_Sensor_Data:
    TBD
Raw_Replica_Vectors:
    TBD
Parameters:
    N_Points_Per_Update
    N_Sensors
    N_Time_Max
    N_FFT_Size
    I_Freq_Bin_First
    I_Freq_Bin_Last
    N_Freq_Bins_Out
    N_Saved_Updates
    N_Freq_Bands
    N_Freq_Bins_Per_Band
    N_Subarrays
    I_First_Sensor[i]   i = 0,...,N_Subarrays - 1
    I_Last_Sensor[i]   i = 0,...,N_Subarrays - 1
    N_AzEl_Beams
    N_AzEl_Beams_Per_Batch
    N_AzEl_Batches
    N_Retained_Times
    N_Replicas
    N_Replicas_Per_Batch
    N_Replica_Batches
    I_B_ME_Flag
    QR_Parameters:
        Inverse_Condition_Number_Threshold
Constraints:
    N_Freq_Bins_Out = I_Freq_Bin_Last - I_Freq_Bin_First + 1

N_FFT_Size = N_Saved_Updates * N_Points_Per_Update
N_Freq_Bins_Out = N_Freq_Bins_Per_Band * N_Freq_Bands
N_AzEl_Beams = N_AzEl_Beams_Per_Batch * N_AzEl_Batches
N_Replicas = N_Replicas_Per_Batch * N_Replica_Batches

A.1.2 Partitioned Array Bartlett Program Input/Outputs

none

A.1.3 Partitioned Array Bartlett Program Outputs

Output_Power:
    TBD
Narrowband_Time_Series:
    TBD

A.1.4 Partitioned Array Bartlett Program Algorithm

```
Read Parameters
Do one-time calculations
Open input and output data files
While more sensor data to read
    Invoke Distribute_to_PEs_by_Sensor process
    Invoke Apply_Windows_and_Perform_FFT process
    While more AzEl batches to read
        Invoke Distribute_AzEl_Vectors process
        Invoke Spatially_Filter_Frequency_Bin_Data process
        Invoke Distribute_and_Sum_NB_Data_by_Subarray process
    End while
    Invoke Factor_Subarray_Matrix process
    While more replicas to read
        Invoke Distribute_Replica_Vectors process
        Invoke Compute_Output_Power_and_Narrowband_Time_Series process
        Invoke Collect_from_PEs_by_Frequency_Band process
    End while
End while
Close input and output data files
```

A.1.5 Partitioned Array Bartlett Program Special Requirements

TBD

A.1.6 Partitioned Array Bartlett Program Validation Criteria

The following tests shall be employed to validate the program:

(i) Simulated acoustic fields arising from two plane waves, together with additive white Gaussian noise, independent and identically distributed from sensor to sensor, shall be generated and supplied as Raw_Sensor_Data. The Output_Power and Narrowband_Time_Series shall be examined for agreement with theoretical predictions. In particular, maximum response should result from those replicas corresponding to the true arrival directions of the plane waves.

(ii) Seatest data shall be processed and the outputs compared with those produced by existing processing software.

A.2.0 Distribute to PEs by Sensor Process

The Distribute to PEs by Sensor Process accesses from mass storage real time series indexed by time and sensor, reorganizes it if necessary, and routes it to PEs. The output data is organized in time updates, one sensor per PE.

A.2.1 Distribute to PEs by Sensor Process Inputs

Raw_Sensor_Data:
  TBD
Parameters:
  N_Points_Per_Update
  N_Sensors
  N_Time_Max
Time_Index:
  I_Time

A.2.2 Distribute to PEs by Sensor Process Input/Outputs

Sensor_History:
  xh[i, j, k] i = 0, ..., N_Points_Per_Update - 1,
         j = 0, ..., N_Sensors - 1
         k = 0, ..., N_Saved_Updates - 1

```
        xh real
    K_Oldest_Update
```

A.2.3 Distribute to PEs by Sensor Process Outputs

none

A.2.4 Distribute to PEs by Sensor Process Algorithm

```
For each j in 0, ..., N_Sensors - 1
    Fill xh[i, j, K_Oldest_Update]
End for
K_Oldest_Update = ( K_Oldest_Update + 1 ) mod N_Saved_Updates
```

A.2.5 Distribute to PEs by Sensor Process Special Requirements

The Sensor_History xh[] shall be 16-bit real.

A.2.6 Distribute to PEs by Sensor Process Validation Criteria

The following test shall be employed to validate the process:

(i) The time index and sensor index are to be encoded into each data value of Raw_Sensor_Data. The Sensor_History values x[i, j, k] shall then be examined for agreement with (i, j).

A.3.0 Apply Windows and Perform FFT Process

The Apply Windows and Perform FFT Process transforms blocks of time series to the frequency domain. A circular buffer of input data is maintained.

A.3.1 Apply Windows and Perform FFT Process Inputs

```
Spectral_Analysis_Window:
    w[i]  i = 0, ..., N_FFT_Size - 1
          w real
Sensor_History:
    xh[i, j, k]  i = 0, ..., N_Points_Per_Update - 1,
                 j = 0, ..., N_Sensors - 1
                 k = 0, ..., N_Saved_Updates - 1
                 xh real
    K_Oldest_Update
Parameters:
```

N_Points_Per_Update
N_Sensors
N_FFT_Size
N_Saved_Updates
N_Time_Max
I_Freq_Bin_First
I_Freq_Bin_Last
N_Freq_Bins_out

Time_Index:
I_Time

A.3.2 Apply Windows and Perform FFT Process Input/Outputs

none

A.3.3 Apply Windows and Perform FFT Process Outputs

Raw_Frequency_Bin_Data:

yr[i, j]   i = 0, ..., N_FFT_Size - 1
j = 0, ..., N_Sensors - 1
yr complex

A.3.4 Apply Windows and Perform FFT Process Algorithm

```
Define C(L) = L mod N_Points_Per_Update
Define D(L) = ( K_Oldest_Update + L / N_Points_Per_Update ) mod N_Saved_Updates
For each j in 0, ..., N_Sensors - 1
      xw[i, j] = w[i] xh[C(i), j, D(i)],
            i = 0, ..., N_FFT_Size - 1
      yr[i, j] = FFT(i; N_FFT_Size; xw[., j]),
            i = 0, ..., N_FFT_Size - 1
End for
```

A.3.5 Apply Windows and Perform FFT Process Special Requirements

The Sensor_History xh[] shall be 16-bit real.

The xw arrays shall be complex so that a complex-to-complex FFT may be used.

A.3.6 Apply Windows and Perform FFT Process Validation Criteria

Tests for validating this process are described in the document "Preliminary Requirements Specification: Function Validation".

A.4.0 Distribute AzEl Vectors Process

The Distribute AzEl Vectors Process routes the steering vectors for Azimuth-Elevation beams so that each PE has the vectors for all frequency bins to be processed, and for the channels which it FFTed.

A.4.1 Distribute AzEl Vectors Inputs

Raw_Azimuth_Elevation_Vectors:
    TBD
Parameters:
    N_Freq_Bands
    N_Freq_Bins_Per_Band
    N_AzEl_Beams_Per_Batch
    N_Sensors

A.4.2 Distribute AzEl Vectors Input/Outputs

none

A.4.3 Distribute AzEl Vectors Outputs

AzEl_Vectors:

	va[i1, i2, k2, j]	i1 = 0, ..., N_Freq_Bands - 1
		i2 = 0, ..., N_Freq_Bins_Per_Band - 1
		k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
		j = 0, ..., N_Sensors - 1
		va complex

A.4.4 Distribute AzEl Vectors Process Algorithm

TBD

A.4.5 Distribute AzEl Vectors Special Requirements

None

A.4.6 Distribute AzEl Vectors Validation Criteria

The following test shall be employed to validate the process:

The sensor number, frequency bin, and AzEl vector number shall be encoded in the Raw_AzEl_Vectors. The AzEl_Vectors shall be examined for agreement with [i1, i2, k2, j].

A.5.0 Spatially Filter Frequency Bin Data Process

The Spatially Filter Frequency Bin Data Process applies the weights of each AzEl vectors for each frequency bin to each sensor.

A.5.1 Spatially Filter Frequency Bin Data Process Inputs

Raw_Frequency_Bin_Data:

	yr[i, j]	i = 0, ..., N_FFT_Size - 1
		j = 0, ..., N_Sensors - 1
		yr complex

AzEl_Vectors:

	va[i1, i2, k2, j]	i1 = 0, ..., N_Freq_Bands - 1
		i2 = 0, ..., N_Freq_Bins_Per_Band - 1
		k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
		j = 0, ..., N_Sensors - 1
		va complex

A.5.2 Spatially Filter Frequency Bin Data Process Input/Outputs

none

A.5.3 Spatially Filter Frequency Bin Data Process Outputs

Raw_Filtered_NB_Data:

fr[i1, i2, k1, k2, j]	i1 = 0, ..., N_Freq_Bands - 1
	i2 = 0, ..., N_Freq_Bins_Per_Band - 1
	k1 = 0, ..., N_AzEl_Batches - 1
	k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
	j = 0, ..., N_Sensors - 1
	fr complex

A.5.4 Spatially Filter Frequency Bin Data Process Algorithm

```
For each i1 in 0, ..., N_Freq_Bands - 1
    For each i2 in 0, ..., N_Freq_Bins_Per_Band - 1
        i = I_Freq_Bin_First + i1*N_Freq_Bins_Per_Band + i2
        For each k2 in 0, ..., N_AzEl_Beams_Per_Batch - 1
            For each j in 0, ..., N_Sensors - 1
                fr[i1, i2, k1, k2, j]
                    = va[i1, i2, k2, j]*yr[i, j]
            End for
        End for
    End for
End for
```

A.5.5 Spatially Filter Frequency Bin Data Process Special Requirements

The index k1 associated with the current AzEl batch is under control of the loop "While more AzEl batches to read".

A.5.6 Spatially Filter Frequency Bin Data Process Validation Criteria

The following test shall be employed to validate the process:

Raw_Frequency_Bin_Data and AzEl_Vectors shall be synthesized such that the real part of the Raw_Filtered_NB_Data will be equal to the sensor number and the imaginary part will be encoded with the frequency bin number and the vector number. The process will be run and the output examined for correctness.

A.6.0 Distribute and Sum NB Data by Subarray Process

The Distribute and Sum NB Data by Subarray Process routes frequency bin data so that each PE has weighted data for selected AzEl beams for all sensors for selected frequency bins. The PEs then sum the weighted sensor data to to form a narrowband time series for each AzEl beam for each subarray.

A.6.1 Distribute and Sum NB Data by Subarray Process Inputs

Raw_Filtered_NB_Data:

fr[i1, i2, k1, k2, j]	i1 = 0, ..., N_Freq_Bands - 1
	i2 = 0, ..., N_Freq_Bins_Per_Band - 1
	k1 = 0, ..., N_AzEl_Batches - 1
	k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
	j = 0, ..., N_Sensors - 1
	fr complex

Parameters:

N_Freq_Bands	
N_Freq_Bins_Per_Band	(= N_Freq_Bins_Out / N_Freq_Bands)
N_AzEl_Beams	
N_AzEl_Batches	
N_AzEl_Beams_Per_Batch	
N_Subarrays	
I_First_Sensor[s]	s = 0,...,N_Subarrays - 1
I_Last_Sensor[s]	s = 0,...,N_Subarrays - 1

Time Index:

I_Time

A.6.2 Distribute and Sum NB Data by Subarray Process Input/Outputs

none

A.6.3 Distribute and Sum NB Data by Subarray Process Outputs

Subarray_AzEl_NB_Time_Series:

yh[i1, i2, k1, k2, s, n]	i1 = 0, ..., N_Freq_Bands - 1

i2 = 0, ..., N_Freq_Bins_Per_Band - 1
k1 = 0, ..., N_AzEl_Batches - 1
k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
s = 0, ..., N_Subarrays - 1
n = 0, ..., N_Retained_Times - 1

A.6.4 Distribute and Sum NB Data by Subarray Process Algorithm

```
Define C(m) = m mod N_Retained_Times
For each i1 in 0, ..., N_Freq_Bands - 1
    For each i2 in 0, ..., N_Freq_Bins_Per_Band - 1
        For each k1 in 0, ..., N_AzEl_Batches - 1
            For each k2 in 0, ..., N_AzEl_Beams_Per_Batch - 1
                For each s in 0, ..., N_Subarrays - 1
                    yh[i1, i2, k1, k2, s, C(I_Time)] = 0
                    For each j in 0, ..., I_First_Sensor[s], ..., I_Last_Sensor[s]
                        yh[i1, i2, k1, k2, s, C(I_Time)] =
                            yh[i1, i2, k1, k2, s, C(I_Time)] + fr[i1, i2, k1, k2, j]
                    End for
                End for
            End for
        End for
    End for
End for
```

A.6.5 Distribute and Sum NB Data by Subarray Process Special Requirements

none

A.6.6 Distribute and Sum NB Data by Subarray Process Validation Criteria

The following test shall be employed to validate the process:

(i) The frequency index, sensor index, and AzEl beam index shall be encoded into each data value of Raw_Filtered_NB_Data. The Subarray_AzEl_NB_Time_Series values yh[i1, i2, k1, k2, s, n] shall then be examined for correctness.

A.7.0 Factor Subarray Matrix Process

The Factor Subarray Matrix Process updates X and performs its QR factorization, where $A = XX^*$ is the cross-spectral matrix.

A.7.1 Factor Subarray Matrix Process Inputs

Subarray_AzEl_NB_Time_Series:

```
    yh[i1, i2, k1, k2, s, n]      i1 = 0, ..., N_Freq_Bands - 1
                                  i2 = 0, ..., N_Freq_Bins_Per_Band - 1
                                  k1 = 0, ..., N_AzEl_Batches - 1
                                  k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
                                  s = 0, ..., N_Subarrays - 1
                                  n = 0, ..., N_Retained_Times - 1;
                                  yh complex
```

Parameters:

```
    N_Sensors
    N_Freq_Bands
    N_Freq_Bins_Per_Band          (= N_Freq_Bins / N_Freq_Bands)
    N_Retained_Times
    N_Time_Max
    N_Subarrays
    N_AzEl_Beams
    N_AzEl_Beams_Per_Batch
    N_AzEl_Batches
    QR_Parameters:
        Inverse_Condition_Number_Threshold
    Time_Index:
    I_Time
```

A.7.2 Factor Subarray Matrix Process Input/Outputs

none

A.7.3 Factor Subarray Matrix Process Outputs

Data_Matrix_Factorization:

```
    to[i1, i2, k1, k2, m, n]; i1 = 0, ..., N_Freq_Bands - 1,
                              i2 = 0, ..., N_Freq_Bins_Per_Band - 1,
                              k1 = 0, ..., N_AzEl_Batches - 1,
                              k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1,
```

```
m = 0, ..., N_Subarrays - 1
n = 0, ..., N_Subarrays - 1
to complex
```

A.7.4 Factor Subarray Matrix Process Algorithm

```
Define C(L) = L mod N_Retained_Times
Define X to be a matrix (N_Subarrays by N_Retained_Times) such that
     X[m, n] = yh[i1, i2, k1, k2, m, n],     m = 0, ..., N_Subarrays - 1,
                                             n = 0, ..., N_Retained_Times - 1
                                             (one such X for each value of
                                             (i1, i2, k1, k2))
Define T to be an upper triangular matrix (N_Subarrays by N_Subarrays)
such that
     T[m, n] = to[i1, i2, k1, k2, m, n],     m = 0, ..., N_Subarrays - 1
                                             n = 0, ..., N_Subarrays - 1
                                             (one such T for each value of
                                             (i1, i2, k1, k2))
If I_B_ME_Flag = 0 then return
If I_Time < N_Retained_Times then return
For each i1 in 0, ..., N_Freq_Bands - 1
     For each i2 in 0, ..., N_Freq_Bins_Per_Band - 1
          For each k1 in 0, ..., N_AzEl_Batches - 1
               For each k2 in 0, ..., N_AzEl_Beams_Per_Batch - 1
                    Matrix computation: T = QR_Factorization(QR_Parameters; X)
               End for
          End for
     End for
End for
```

A.7.5 Factor Subarray Matrix Process Special Requirements

The inverse condition number shall be monitored; if it falls below Inverse_Condition_Number_Threshold, a diagnostic message shall be produced.

A.7.6 Factor Subarray Matrix Process Validation Criteria

The matrices T, X should satisfy the condition

$$TT^* = XX^*$$

where * denotes conjugate transpose.

A.8.0 Distribute Replica Vectors Process

The Distribute Replica Vectors Process accesses replicas from mass storage and routes them to appropriate PEs.

A.8.1 Distribute Replica Vectors Process Inputs

Raw_Replica_Vectors:
TBD
Parameters:

N_Freq_Bands
N_Freq_Bins_Per_Band
N_AzEl_Batches
N_AzEl_Beams_Per_Batch
N_Subarrays
N_Retained_Times
N_Replicas
N_Replicas_Per_Batch
N_Replica_Batches

A.8.2 Distribute Replica Vectors Process Input/Outputs

TBD

A.8.3 Distribute Replica Vectors Process Outputs

Replica_Vectors:

vi[i1, i2, k1, k2, r, s],	i1 = 0, ..., N_Freq_Bands - 1
	i2 = 0, ..., N_Freq_Bins_Per_Band - 1
	k1 = 0, ..., N_AzEl_Batches - 1
	k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
	r = 0, ..., N_Replicas_Per_Batch - 1
	s = 0, ..., N_Subarrays - 1
	vi complex

A.8.4 Distribute Replica Vectors Process Algorithm

TBD

A.8.5 Distribute Replica Vectors Process Special Requirements

TBD

A.8.6 Distribute Replica Vectors Process Validation Criteria

TBD

A.9.0 Compute Output Power and Narrowband Time Series Process

The Compute Output Power and Narrowband Time Series Process forms and outputs either Bartlett or Minimum Energy power for a set of input replica vectors.

A.9.1 Compute Output Power and Narrowband Time Series Process Inputs

Subarray_AzEl_NB_Time_Series:	
yh[i1, i2, k1, k2, s, n]	i1 = 0, ..., N_Freq_Bands - 1
	i2 = 0, ..., N_Freq_Bins_Per_Band - 1
	k1 = 0, ..., N_AzEl_Batches - 1
	k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
	s = 0, ..., N_Subarrays - 1
	n = 0, ..., N_Retained_Times - 1;
	yh complex
Data_Matrix_Factorization:	
to[i1, i2, k1, k2, m, n],	i1 = 0, ..., N_Freq_Bands - 1,
	i2 = 0, ..., N_Freq_Bins_Per_Band - 1,
	k1 = 0, ..., N_AzEl_Batches - 1,
	k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1,
	m = 0, ..., N_Subarrays - 1
	n = 0, ..., N_Subarrays - 1
	to complex
Replica_Vectors:	
vi[i1, i2, k1, k2, r, s],	i1 = 0, ..., N_Freq_Bands - 1
	i2 = 0, ..., N_Freq_Bins_Per_Band - 1
	k1 = 0, ..., N_AzEl_Batches - 1
	k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
	r = 0, ..., N_Replicas_Per_Batch - 1
	s = 0, ..., N_Subarrays - 1
	vi complex

Parameters:
  N_Freq_Bands
  N_Freq_Bins_Per_Band
  N_AzEl_Batches
  N_AzEl_Beams_Per_Batch
  N_Subarrays
  N_Retained_Times
  N_Replicas_Per_Batch
  I_B_ME_Flag
Time_Index:
  I_Time

A.9.2 Compute Output Power and Narrowband Time Series Process Input/Outputs

A.9.3 Compute Output Power and Narrowband Time Series Process Outputs

Raw_Output_Power:

p[i1, i2, k1, k2, r],	i1 = 0, ..., N_Freq_Bands - 1
	i2 = 0, ..., N_Freq_Bins_Per_Band - 1
	k1 = 0, ..., N_AzEl_Batches - 1
	k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
	r = 0, ..., N_Replicas_Per_Batch - 1
	p real

Raw_Narrowband_Time_Series:
  TBD

A.9.4 Compute Output Power and Narrowband Time Series Process Algorithm

Define v to be a vector (length N_Subarrays) such that

v[s] = vi[i1, i2, k1, k2, r, s]	s = 0, ..., N_Subarrays - 1
	(one such v for each value of (i1, i2, k1, k2, r))

Define X to be a matrix (N_Subarrays by N_Retained_Times) such that

X[m, n] = yh[i1, i2, k1, k2, m, n]	m = 0, ..., N_Subarrays - 1

```
                                           n = 0, ..., N_Retained_Times - 1
                                           (one such X for each value
                                           of (i1, i2, k1, k2))
Define T to be an upper triangular matrix (N_Subarrays by N_Subarrays)
such that
    T[m, n] = to[i1, i2, k1, k2, m, n]     m = 0, ..., N_Subarrays - 1
                                           n = 0, ..., N_Subarrays - 1
                                           (one such T for each value
                                           of (i1, i2, k1, k2))
If I_Time < N_Retained_Times and I_B_ME_Flag = 1 then return

For each i1 in 0, ..., N_Freq_Bands - 1
  For each i2 in 0, ..., N_Freq_Bins_Per_Band - 1
    For each k1 in 0, ..., N_AzEl_Batches - 1
      For each k2 in 0, ..., N_AzEl_Beams_Per_Batch - 1
        For each r in 0, ..., N_Replicas_Per_Batch - 1
          Switch on I_B_ME_Flag
            Case 0:
              Matrix computation: w = Xv*
              Matrix computation: p[i1, i2, k1, k2, r] = w*w
            End case 0
            Case 1:
              Matrix computation: w = Backsolve(T;v)
              Matrix computation: p[i1, i2, k1, k2, r] = (w*w)^-1
            End case 1
          End switch
        End for
      End for
    End for
  End for
End for
```

A.9.5 Compute Output Power and Narrowband Time Series Process Special Requirement

A.9.6 Compute Output Power and Narrowband Time Series Process Validation Criteria

A.10.0 Collect from PEs by Frequency Band Process

The Collect from PEs by Frequency Band Process routes output power and narrowband time series for output to mass storage.

A.10.1 Collect from PEs by Frequency Band Process Inputs

Raw_Output_Power:

p[i1, i2, k1, k2, r], i1 = 0, ..., N_Freq_Bands - 1
i2 = 0, ..., N_Freq_Bins_Per_Band - 1
k1 = 0, ..., N_AzEl_Batches - 1
k2 = 0, ..., N_AzEl_Beams_Per_Batch - 1
r = 0, ..., N_Replicas_Per_Batch - 1
p real

Raw_Narrowband_Time_Series:
TBD

Parameters:
N_Freq_Bands
N_Freq_Bins_Per_Band
N_AzEl_Batches
N_AzEl_Beams_Per_Batch
N_Replicas_Per_Batch

A.10.2 Collect from PEs by Frequency Band Process Input/Outputs

TBD

A.10.3 Collect from PEs by Frequency Band Process Outputs

Output_Power:
TBD

Narrowband_Time_Series:
TBD

A.10.4 Collect from PEs by Frequency Band Process Algorithm

TBD

A.10.5 Collect from PEs by Frequency Band Process Special Requirements

TBD

A.10.6 Collect from PEs by Frequency Band Process Validation Criteria

TBD

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1991	Final: Dec 1988 — July 1991

4. TITLE AND SUBTITLE

MATCHED FIELD PROCESSING ON THE CONNECTION MACHINE

5. FUNDING NUMBERS

PE: 62234N
PROJ: RS34J77
TASK: 01
WU: DN300086

6. AUTHOR(S)

T. A. Adams

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
San Diego, CA 92152-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NOSC TR 1440

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Technology (Code 227)
800 N. Quincy
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Matched field processing was implemented on the Thinking Machine Corporation's Connection Machine (model CM-2), which employs thousands of bit-serial processors. The potential of the CM-2 was not realized in this initial implementation because the programming style and language features used led to a large interprocessor communications burden. In subsequent efforts at developing signal processing on parallel processors, there should be additional emphasis on decomposing the overall processing into a relatively small set of building blocks that are of higher level than elementary arithmetic operations on scalars.

14. SUBJECT TERMS

matched field processing (MFP)
sonar
propagation
software portability
Connection Machine
ocean acoustics
signal processing
parallelism

15. NUMBER OF PAGES

41

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
T. A. Adams	(619) 553–5990	Code 733

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0142	K. J. Campbell	(1)
Code 0144	R. November	(1)
Code 402	R. A. Wasilausky	(1)
Code 7104	R. H. Hearn	(1)
Code 7304	G. L. Mohnkern	(1)
Code 7304	W. H. Marsh	(1)
Code 733	T. A. Adams	(2)
Code 733	R. W. Myers	(1)
Code 733	D. F. Schwartz	(1)
Code 761	S. I. Chou	(1)
Code 761	C. V. Tran	(1)
Code 952B	J. Puleo	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(3)

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

ODDRE (R&AT)/SCT
Washington, DC 20301-3080

NOSC Liaison Office
Washington, DC 20363-5100

Navy Acquisition, Research & Development Information Center (NARDIC)
Alexandria, VA 22333

Office of Naval Technology
Arlington, VA 22217-5000

Naval Sea Systems Command
Washington, DC 20362-5101

Naval Underwater Systems Center
Newport, RI 02841-5049 (2)

Rome Air Development Center/COE
Griffiss AFB, NY 13441

XonTech, Inc.
Van Nuys, CA 91406

Defense Advanced Research Projects Agency
Arlington, VA 22209 (2)

Chief of Naval Operations
Washington, DC 20301-3080

Center for Naval Analyses
Alexandria, VA 22302-0268

Navy Acquisition, Research & Development Information Center (NARDIC)
Pasadena, CA 91106-3955

Space & Naval Warfare Systems Command
Washington, DC 20363-5100 (2)

Naval Research Laboratory
Washington, DC 20375-5000 (4)

Naval Weapons Center
China Lake, CA 93555-6001

AMSEL-RD-SE-AST
Fort Monmouth, NJ 07703-5000